6.3. Package Management Linux From Scratch - Version 6.4 Chapter 6. Installing Basic System Software Prev Preparing Virtual Kernel File Systems Next Entering the Chroot Environment Up Home 6.3. Package Management Package Management is an often requested addition to the LFS Book. A Package Manager allows tracking the installation of files making it easy to remove and upgrade packages. As well as the binary and library files, a package manager will handle the installation of configuration files. Before you begin to wonder, NO—this section will not talk about nor recommend any particular package manager. What it provides is a roundup of the more popular techniques and how they work. The perfect package manager for you may be among these techniques or may be a combination of two or more of these techniques. This section briefly mentions issues that may arise when upgrading packages. Some reasons why no package manager is mentioned in LFS or BLFS include: Dealing with package management takes the focus away from the goals of these books—teaching how a Linux system is built. There are multiple solutions for package management, each having its strengths and drawbacks. Including one that satisfies all audiences is difficult. There are some hints written on the topic of package management. Visit the Hints Project and see if one of them fits your need. 6.3.1. Upgrade Issues A Package Manager makes it easy to upgrade to newer versions when they are released. Generally the instructions in the LFS and BLFS Book can be used to upgrade to the newer versions. Here are some points that you should be aware of when upgrading packages, especially on a running system. If one of the toolchain packages (Glibc, GCC or Binutils) needs to be upgraded to a newer minor version, it is safer to rebuild LFS. Though you may be able to get by rebuilding all the packages in their dependency order, we do not recommend it. For example, if glibc-2.2.x needs to be updated to glibc-2.3.x, it is safer to rebuild. For micro version updates, a simple reinstallation usually works, but is not guaranteed. For example, upgrading from glibc-2.3.4 to glibc-2.3.5 will not usually cause any problems. If a package containing a shared library is updated, and if the name of the library changes, then all the packages dynamically linked to the library need to be recompiled to link against the newer library. (Note that there is no correlation between the package version and the name of the library.) For example, consider a package foo-1.2.3 that installs a shared library with name libfoo.so.1. Say you upgrade the package to a newer version foo-1.2.4 that installs a shared library with name libfoo.so.2. In this case, all packages that are dynamically linked to libfoo.so.1 need to be recompiled to link against libfoo.so.2. Note that you should not remove the previous libraries until the dependent packages are recompiled. 6.3.2. Package Management Techniques The following are some common package management techniques. Before making a decision on a package manager, do some research on the various techniques, particularly the drawbacks of the particular scheme. 6.3.2.1. It is All in My Head! Yes, this is a package management technique. Some folks do not find the need for a package manager because they know the packages intimately and know what files are installed by each package. Some users also do not need any package management because they plan on rebuilding the entire system when a package is changed. 6.3.2.2. Install in Separate Directories This is a simplistic package management that does not need any extra package to manage the installations. Each package is installed in a separate directory. For example, package foo-1.1 is installed in /usr/pkg/foo-1.1 and a symlink is made from /usr/pkg/foo to /usr/pkg/foo-1.1. When installing a new version foo-1.2, it is installed in /usr/pkg/foo-1.2 and the previous symlink is replaced by a symlink to the new version. Environment variables such as PATH, LD_LIBRARY_PATH, MANPATH, INFOPATH and CPPFLAGS need to be expanded to include /usr/pkg/foo. For more than a few packages, this scheme becomes unmanageable. 6.3.2.3. Symlink Style Package Management This is a variation of the previous package management technique. Each package is installed similar to the previous scheme. But instead of making the symlink, each file is symlinked into the /usr hierarchy. This removes the need to expand the environment variables. Though the symlinks can be created by the user to automate the creation, many package managers have been written using this approach. A few of the popular ones include Stow, Epkg, Graft, and Depot. The installation needs to be faked, so that the package thinks that it is installed in /usr though in reality it is installed in the /usr/pkg hierarchy. Installing in this manner is not usually a trivial task. For example, consider that you are installing a package libfoo-1.1. The following instructions may not install the package properly: ./configure --prefix=/usr/pkg/libfoo/1.1 make make install The installation will work, but the dependent packages may not link to libfoo as you would expect. If you compile a package that links against libfoo, you may notice that it is linked to /usr/pkg/libfoo/1.1/lib/libfoo.so.1 instead of /usr/lib/libfoo.so.1 as you would expect. The correct approach is to use the DESTDIR strategy to fake installation of the package. This approach works as follows: ./configure --prefix=/usr make make DESTDIR=/usr/pkg/libfoo/1.1 install Most packages support this approach, but there are some which do not. For the non-compliant packages, you may either need to manually install the package, or you may find that it is easier to install some problematic packages into /opt. 6.3.2.4. Timestamp Based In this technique, a file is timestamped before the installation of the package. After the installation, a simple use of the find command with the appropriate options can generate a log of all the files installed after the timestamp file was created. A package manager written with this approach is install-log. Though this scheme has the advantage of being simple, it has two drawbacks. If, during installation, the files are installed with any timestamp other than the current time, those files will not be tracked by the package manager. Also, this scheme can only be used when one package is installed at a time. The logs are not reliable if two packages are being installed on two different consoles. 6.3.2.5. Tracing Installation Scripts In this approach, the commands that the installation scripts perform are recorded. There are two techniques that one can use: The LD_PRELOAD environment variable can be set to point to a library to be preloaded before installation. During installation, this library tracks the packages that are being installed by attaching itself to various executables such as cp, install, mv and tracking the system calls that modify the filesystem. For this approach to work, all the executables need to be dynamically linked without the suid or sgid bit. Preloading the library may cause some unwanted side-effects during

installation. Therefore, it is advised that one performs some tests to ensure that the package manager does not break anything and logs all the appropriate files. The second technique is to use strace, which logs all system calls made during the execution of the installation scripts. 6.3.2.6. Creating Package Archives In this scheme, the package installation is faked into a separate tree as described in the Symlink style package management. After the installation, a package archive is created using the installed files. This archive is then used to install the package either on the local machine or can even be used to install the package on other machines. This approach is used by most of the package managers found in the commercial distributions. Examples of package managers that follow this approach are RPM (which, incidentally, is required by the Linux Standard Base Specification), pkg-utils, Debian's apt, and Gentoo's Portage system. A hint describing how to adopt this style of package management for LFS systems is located at http://www.linuxfromscratch.org/hints/downloads/files/fakeroot.txt. 6.3.2.7. User Based Management This scheme, unique to LFS, was devised by Matthias Benkmann, and is available from the Hints Project. In this scheme, each package is installed as a separate user into the standard locations. Files belonging to a package are easily identified by checking the user ID. The features and shortcomings of this approach are too complex to describe in this section. For the details please see the hint at http://www.linuxfromscratch.org/hints/downloads/files/more_control_and_pkg_man.txt. Prev Preparing Virtual Kernel File Systems Next Entering the Chroot Environment Up Home